Jay Kalyanbhai Savani

20009207

Assignment 2

Chapter – 2

Graphical user interface, text, application

Description automatically generated

Ans: Check if Queue Q1 is empty before using push() on the two Queues, Q1 and Q2. Start enqueuing elements if the queue is empty. In this case, we'll add elements 0, 1, and 2.

Now when we must pop() an element, rule 2 states that it should pop out first. To do this, we'll use Queue Q2 and begin enqueueing elements from Queue Q1. We'll keep doing this until Queue Q1 has just 1 element left, or 2.

We dequeue it now that there are just 2 items in Queue Q1.

We now wish to add element 3. So., we add it to Queue 2.

Push() and pop() operations are carried out in a stack in this manner. This means that two queues can be used to implement a stack.

Enqueue and dequeue operations have an O(1) running time. Push() and pop() operations will therefore likewise operate in O(1) time.

Graphical user interface, text, application, email

Description automatically generated

Ans: We can create a recursive technique to print each node's depth. The depth is 0 when the current node is the tree's root. The depth of its children, which is 1, can then be printed. The depth of their offspring, which is 2, can then be printed. After that, we can print the depth of all the nodes using the recursion method.

Algorithm:

depthOfTree\_T(T,v):

if T.isRoot(v) then

return 0

else

d = 0

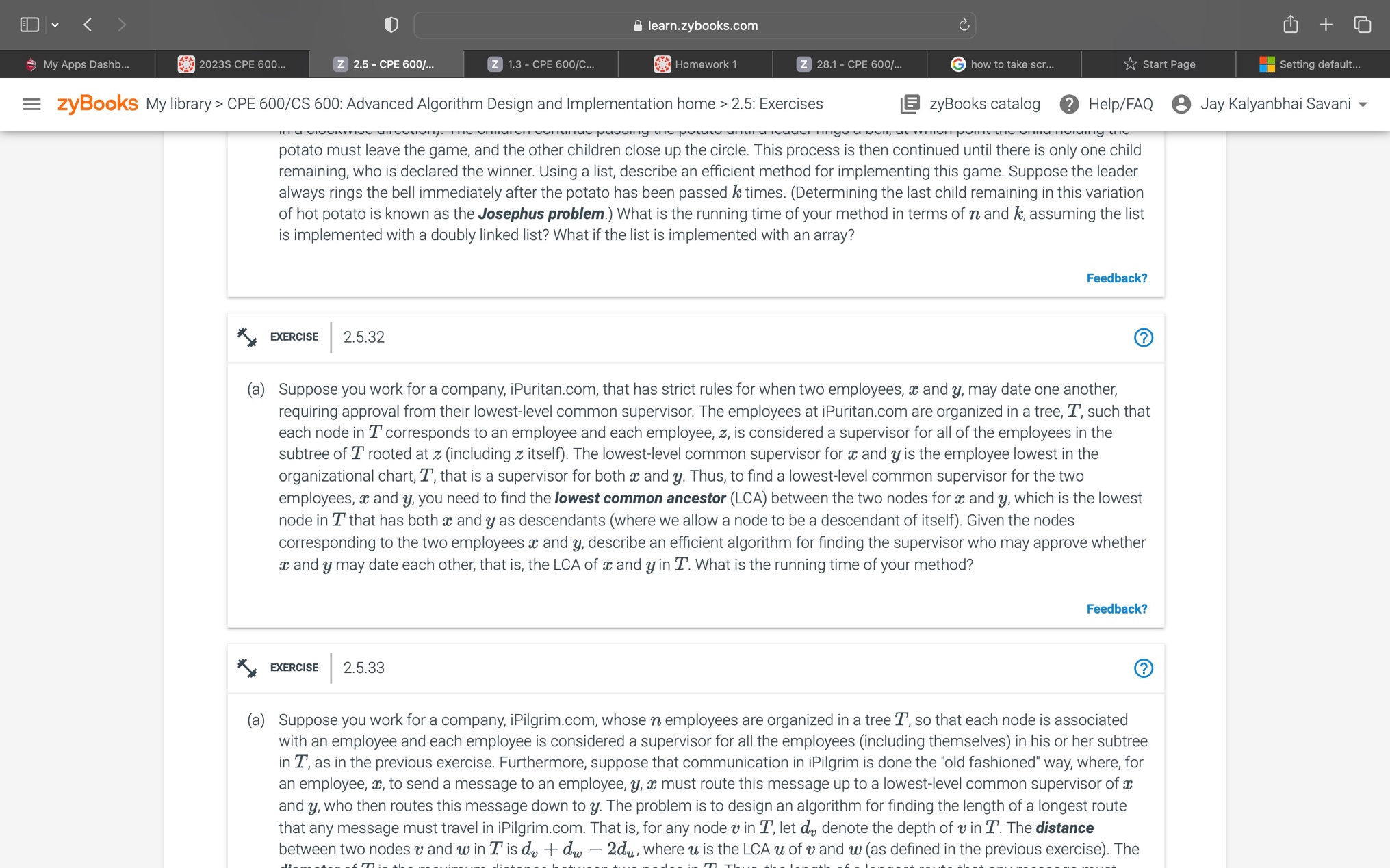
for each w which is a child of v do

d = 1 + depth(T, T.parent(v))

return d

Run time=O(n)

n=number of nodes in a tree.



Ans: Consider the root node and begin traversing it to get the lowest common ancestor between nodes x and y in a tree T. The root of a node is the lowest common ancestor if the specified values for x and y match. Call the lowest common ancestor algorithm iteratively for the left subtree and right subtree if x and y don't match. The parent node is lca if x and y are present as the left child and right child, respectively. If x and y are present in the left subtree, lca is from the left subtree, and vice versa for the right subtree.

Algorithm:

LowestCommonAncestor(BinaryTree x, BinaryTree y, BinaryTree root):

Input: A BinaryTree node root T, a node x and a node y

Output: The lowest common ancestor of x and y

If root is null or x is root or y is root:

Return root

BinaryTree leftNode <- LowestCommonAncestor(x, y, root.left)

BinaryTree rightNode <- LowestCommonAncestor(x, y, root.right)

If leftNode = null:

Return rightNode

Else if rightNode = null:

Return leftNode

Return root

Run time=O(n)

n=height of the tree.

Chapter – 3

Graphical user interface, text, application, email

Description automatically generated

Ans: In a union of keys from S and T, where S and T are both ordered arrays of n items each, one must locate the kth smallest key.

Look at the k/2 member in the array list S first.

Now perform a binary search on the element in T that is the greatest and less than k/2. Add these two items' indices now:

• Take no more than two items if the sum of them, or k, is equal.

• Binary search is carried out to the right of S if sum > k.

• A binary search is conducted to the left of S if sum k.

The identical operations are now carried out on T based on the largest element in S that is less than the element being used right now.

Performing a binary search for two arrays S and T will take O(log n) and O(log n) respectively. That is O(log2 n).  
Solving this will give O(log n) running time complexity.

Graphical user interface, text, application, email

Description automatically generated

Ans: Perform post order traversal on the tree, then recursively call the left and right subtrees and free the nodes in order to remove every node from a binary search tree.

Algorithm:

removeAllElements(target, root):

Input: A search key k for node of a binary search tree T.

Output: Empty binary search tree

if T(k, T.root()) is null

return null

else

removeAllElements( binaryPostorder(T, T.leftChild(k)))

removeAllElements(binaryPostorder(T, T.rightChild(k)))

perform the “free” action for key (k) node

We spend O(h) time to find the node that equals to the target.  
And then we spend s to remove all the duplicate. So, the total time complexity is O(h + s)

Graphical user interface, text, application, email

Description automatically generated

Ans: Using the binary search (division and conquer strategy) to store medicine in the xi millimeter-tiniest bottle in the inventory. The smallest bottle will be found on the left and the largest on the right, as ordered in an array T by capacity.

Diagram, schematic

Description automatically generated

The above approach will give the recurrence relation:

T(n) = T(n/2) + c

Solving this recurrence relation using the iteration method

T(n) = (c + c + T(n/4))

After solving will give

T(n) = k\*c + T(n/2k)

T(n) = c log n

For n requests, the time complexity of the above method is c log n but we must process the drug order of k requests will change the time complexity to O(k log n).

Since the order is already sorted for xi it will take less time to search for k requests.

After every xi which changes the complexity to O(k log n/k).

Chapter – 4

Graphical user interface, text, application, email

Description automatically generated

Ans:

Chart, bubble chart

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Given**: F0 =0 and F1 = 1, and Fk= Fk−1 + Fk−2, for k ≥ 2

**Proof:** Fk ≥ φk−2,

Base Case: for k=2

F2 = F1 + F0 = 1

Fk ≥ φk−2 => φ0 = 1

Therefore, true for k = 2

Base Case: for k=2

F3 = F2 + F1 = 2

Fk ≥ φk−2 => φ

Fk > φ

Therefore, true for k = 3

Now check for k-1

Fk−1 = Fk-2 + Fk−3 > φk−4 + φk−5 = = = (Given: φ2 =φ+1 )

Fk−1 ≥ φk−3 (Assumption true for k-1)

So, it will be true for Fk ≥ φk−2

Hence, we can say that, for k >= 3, Fk >= φk−2

**Graphical user interface, text, application

Description automatically generated**

Ans: Bin packing problem: A finite number of bins or containers, each of volume V, must be filled with objects of varied volumes in a method that uses the fewest possible bins. Initial fit heuristic The objects are processed by the algorithm in any sequence. It makes an effort to put each thing in the first bin that has room for it. If no bin is present, a new bin is opened and the object is placed inside of it. We can reduce the running time complexity of saving the photos onto the hard drives from O(mn) to O with the aid of balancing search trees (AVL tree) (m log n). n m is the number of USB drives, and m is the number of photos. As opposed to checking putting images in an order to check all the m drives to see whether there is any space remaining in the previous bins (according to First fit), which takes O(log n) for n items, executing insertion operation in AVL tree takes m running time. Therefore, the initial fit's total running time complexity is O. (m log n).